Minarea regulilor de asociere

Algoritmul Apriori

Stîngă Alexandra

Introducere

Data mining este procesul de descoperire a patternurilor în seturi de date de mari dimensiuni folosind modalități de machine learning, statistică și baze de date. Data mining, mineritul de date în traducere liberă, are rolul de a extrage în mod inteligent informația dintr-un set de date și să o transforme într-o structură ulterior folosită în luarea deciziilor.

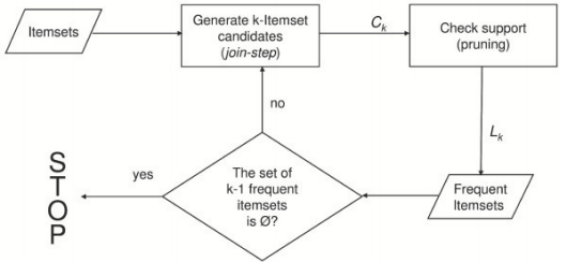
Un algoritm clasic de data mining este algoritmul Apriori. Acesta a fost propus de Agrawal și Srikant in 1994 și este folosit pentru minarea itemseturilor frecvente și generarea regulilor de asociere.

Reguli de asociere. Exemplu

Algoritmul apriori este folosit în găsirea itemseturilor frecvente într-o bază de date pentru reguli de asociere booleene. Numele algoritmului este Apriori deoarece se folosește de cunoaște dinainte proprietățile itemseturilor.

Acest algoritm constă într-o căutare iterativă în care k-itemseturile frecvente sunt folosite pentru a găsi (k+1)-itemseturile frecvente.Algoritmul începe prin determinarea 1-itemseturilor frecvente.Pentru aceasta se scanează baza de date și se determină suportul(numărul de apariții) fiecarui element din BD,care se va compara cu suportul minim. Itemseturile alese vor fi cele din care se vor genera 2-itemseturile, tot așa până când mulțimea generală k-itemset este vidă.

Figure 1: Schema logică a algoritmului Apriori



Pentru a mări eficiența generării itemseturilor frecvente se folosesc următoarele proprietăți: toate subseturile unui itemset frecvent trebuie să fie frecvente și dacă un itemset este infrecvent atunci toate superseturile vor fi infrecvente.

În continuare este prezentat un exemplu practic folosind algoritmul descris mai sus. Datele de input sunt baza de date cu tranzacțiile dintr-un magazin, suportul minim care este 2 și confidența 4.

Table 1: The dataset.

ID	Transaction
1	unt, oua, lapte
2	paine, unt
3	paine, oua, lapte
4	paine, unt, oua, lapte
5	oua, lapte, salam
6	paine, oua

Calculăm frecvența fiecărui itemset prin scanări repetate ale bazei de date.

Table 2: The items and their frequence.

Itemset	Frequence
unt	3
oua	5
lapte	4
paine	4
salam	1

Comparând frecvența fiecăruia cu suportul minim obținem 1-itemset.

Table 3: The items and their frequence.

Itemset	Frequence
unt	3
oua	5
lapte	4
paine	4

Itemsetul de k = 2 valori este obținut prin unirea listei anterioare de itemseturi cu ea însuși. Condiția de unire este aceea ca cele 2 liste să aibă k-2 elemente în comun.

Se verifică dacă toate subseturile unui itemset sunt frecvente sau nu. În cazul în care itemsetul nu este frecvent va fi elliminat.

De exemplu subseturile itemsetului [unt, oua] sunt [unt] și [ouă] care sunt frecvente.

La fiecare pas se calculează suportul fiecărui itemset parcurgând baza de date. Dacă suportul itemsetului este mai mare decât suportul minim, atunci

itemsetul va face parte din lista finală de itemseturi din care vor fi generate k+1 itemseturi.

Table 4: The itemsets and their frequence.

Itemset	Frequence
unt, oua	2
unt, lapte	2
unt, paine	2
oua, lapte	4
oua, paine	3
lapte, paine	2

Itemsetul de 3 valori este:

Table 5: The itemsets and their frequence.

Itemset	Frequence
unt, oua, lapte	2
oua, paine, paine	2

Aici algoritmul se oprește deoarece nu se mai pot genera itemseturi de lungime 4. După ce s-au descoperit toate itemseturile frecvente, se trece la generarea regulilor de asociere. Pentru asta este nevoie de calcularea confidenței fiecărei reguli.

O regulă de asociere arată în felul următor:

$$Confidence(A->B) = SupportCount(A \cup B)/SupportCount(A) \quad (1)$$

Itemsetul [unt, oua, lapte] are următoarele reguli de asociere:
[unt -> ouă] => lapte <=> support(unt,ouă, lapte)/support(unt,ouă) = 2/2 = 100
[unt -> lapte] => ouă <=> support(unt,ouă, lapte)/support(unt,lapte) = 2/2 = 100
[ouă -> lapte] => unt <=> support(unt,ouă, lapte)/support(ouă,lapte) = 2/4 = 50
[unt] -> [ouă -> lapte] <=> support(unt,ouă, lapte)/support(unt) = 2/3 = 66
[ouă] -> [unt -> lapte] <=> support(unt,ouă, lapte)/support(ouă) = 2/5 = 40
[lapte] -> [unt -> ouă] <=> support(unt,ouă, lapte)/support(lapte) = 2/4 = 50

Algoritmul apriori. Cod

Aplicația practică se folosește de baza de date din primul tabel și generează regulile de asociere corespunzătoare.

Baza de date este încărcată în memorie, într-o structură de date de tip listă de Stringuri, pentru folosiri ulterioare.

```
//read database
private ArrayList<String> readDB() {

     ArrayList<String> tranzactions = new ArrayList<>();
     BufferedReader br = null;
     String st = null;

   try {
      br = new BufferedReader(new FileReader(file));
   } catch (FileNotFoundException e1) {
      // TODO Auto-generated catch block
      e1.printStackTrace();
   }

    try {
      while ((st = br.readLine()) != null) {
         tranzactions.add(st);
         nrTranzactii++;
         }
      } catch (IOException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

      return tranzactions;
   }
```

În primul pas al algoritmului se determină itemseturile frecvente de k=1. Se parcurge lista de tranzacții si se stochează suportul fiecărui item în tabela newCandidates. După terminarea acestui pas se aleg itemurile care depășesc suportul minim și se trece la generarea itemseturilor de k+1.

```
//generate itemset of size 1
private ArrayList<String> itemsetSize1(ArrayList<String> candidates){

      LinkedHashMap<String, Integer> newCandidates = new
          LinkedHashMap<>();
      ArrayList<String> itemsList = new ArrayList<>();

      Iterator<String> li = tranzactions.iterator();
      String tmpString;
      List<String> trans_items;

      //init the candidates hashmap
      int count = 0;
      candidates.forEach(candidat -> newCandidates.put(candidat,
          count));
//    candidates.forEach(i -> System.out.println(i));

      while(li.hasNext()) {
         //for each transaction
         tmpString = li.next();
         trans_items = Arrays.asList(tmpString.split(",[ ]*"));
```

```
        //update the counter of each item
        for(int i = 0; i<trans_items.size(); i++) {
           newCandidates.put(trans_items.get(i),
               newCandidates.get(trans_items.get(i)) + 1);
        }
     }

     //check if the frecquence of each item > minim_support
     for (Entry<String, Integer> m: newCandidates.entrySet()) {
        if(m.getValue() >= minSup )
           itemsList.add(m.getKey());
       }

     //Sysout chosen items
     for (Entry<String, Integer> m: newCandidates.entrySet()) {
        System.out.println(m.getKey()+" "+m.getValue());
     }

     return itemsList;
   }
```

După ce s-a determinat suportul fiecărui produs, se poate trece la generarea itemseturilor de 2,3,... etc.

```
//generate itemset of size N
private ArrayList<ArrayList<String>>
    generateNitems(ArrayList<ArrayList<String>> items, int cicle){

     System.out.println("Cicle "+cicle);
     ArrayList<ArrayList<String>> newItems = new ArrayList<>();
     ArrayList<String> tmpSet1,tmpSet2, tmpItem = new ArrayList<>();

     int i,j;
     for(i = 0; i < items.size(); i++) {
        for(j = i+1;j < items.size(); j++) {

           /*get the 2 list*/
           tmpSet1 = items.get(i);
           tmpSet2 = items.get(j);

           /*det if are cicle-1 elem common*/
           if(canJoin(tmpSet1, tmpSet2,cicle)) {
              tmpItem = joinList(tmpSet1, tmpSet2);

              /*det if has frequent subsets*/
              if(hasFrequent(items,tmpItem,cicle))
                 newItems.add(tmpItem);
           }
        }
     }

//   System.out.println("generateNitems "+newItems.toString());
     return newItems;
```

```
  }
```

Generarea regulilor ded asociere se face după formula prezentată mai sus.

```
//assoc rules
private void generateAssocRules(ArrayList<ArrayList<String>>
    assocCandidates) {

      System.out.println("Association rules");
      assocCandidates.forEach(i -> System.out.println(i));
      int pairCount;

      LinkedHashMap<String, Integer> onesHashMap = new
          LinkedHashMap<>();
      for(int i=0;i<assocCandidates.size();i++) {
         assocCandidates.get(i).forEach(candidat ->
             onesHashMap.put(candidat, 0));
      }

      //count itemset appearences
      countOne(onesHashMap);

      if(assocCandidates.get(0).size() == 2) {
         for(int index =0;index < assocCandidates.size(); index++) {
            pairCount = countPair(assocCandidates.get(index));
            associationsHashMap.put(assocCandidates.get(index),
                ((float)
                pairCount/onesHashMap.get(assocCandidates.get(index).get(0))));
         }
      }
      for (Entry<ArrayList<String>, Float> m:
          associationsHashMap.entrySet()) {
         System.out.println(m.getKey()+" "+m.getValue());
      }

  }
```